基于 Spark 的 SKA1-MID 自校准管线分布计算实现

戴伟[1] 汪森[1] 李秋虹[2] 邓辉[3] 梅盈[3] 王锋[13*]

1.昆明理工大学云南省计算机技术应用重点实验室，云南昆明，650051

2.复旦大学，上海，210000

3.广州大学天体物理中心,广东广州，510006

摘 要

SKA 科学数据处理产生的数据超出了所有已存在的分布式处理系统的处理能力，如何实现一个分布式执行框架是当前科学数据处理的一个重要研究内容。Spark 是非常成熟的一个商业框架，在互联网应用中被广泛应用，本文根据 SKA 项目进展要求，重点研究了如何将算法参考库(ARL)中的部分管线移植到 Spark 上执行。本文对部分实现过程进行了分析讨论，给出了相应的任务流程实现。最终结果表明，移植后代码生成结果符合预期，Spark 能够满足部分数据分布式数据的要求，但迫切需要解决自身存在的一系列问题。

关键词：分布计算管线, 算法参考库-ARL，分布数据处理

中图分类号： TP3

一、前言

为了进一步开展当前重大科学问题如暗物质和暗能量、黑洞和致密天体、宇宙起源、天体起源以及宇宙生命起源等研究工作的需要，平方公里阵（Square Kilometre Array, SKA）项目[1, 2]被提出，并成为国际上即将建造的最大综合孔径射电望远镜[3]，得到全球天文学家的重点关注。SKA 建在澳大利亚、南非及南部非洲 8 个国家的无线电宁静区，其接收面积可达 1 平方公里，频率覆盖 50MHz-20GHz。SKA 作为下一代射电望远镜，具有极高的灵敏度（比 JVLA 灵敏度提高 50 倍,搜寻速度提高 10000 倍）, 以千公里的基线获得极高的空间分辨率，以纳秒级的采样获得精细的时间结构，以 10Pb/s 的速率产生超越全球互联网总量的数据。以宽视场、多波束、高动态、高分辨和大数据为核心概念的 SKA 将颠覆射电天文学的传统研究手段，给天文学研究带来革命性全新的理念。

SKA 高分辨率、高灵敏度、高动态范围和宽视场等新特性，给 SKA 数据处理特别是工业界带来了前所未有的挑战。SKA 的巨大规模和复杂程度远远超出了现有射电天文望远镜阵列，全规模运行的 SKA 产生的海量数据需要 10 亿亿次/秒处理能力，是目前世界上最快的超级计算机神威太湖之光处理能力（0.9 亿亿次/秒）的 10 倍。考虑到计算效率和软件执行效率（目前天文软件在超算平台上的执行效率普遍在 10%甚至更低）, 实际需求将大大超出这个理论估算[4]。

为了充分利用计算资源，确保数据处理流程的可靠性，近 5 年来，SKA 的科学数据处理器（SDP）一直在研究与测试执行框架(Execution Framework)技术，以期找到满足未来发展要求，性能突出的执行框架技术。这其中，除了本文作者所在项目组参与的 DALiuGE 等相关工作[5, 6]以外，也一直在研讨商用执行框架如 Spark[7, 8]、DASK[9]的可用性。本文的工作，正是针对这一方面的工作，细致讨论了 SPARK 执行框架在未来 SKA 科学数据处理

---

[1] 基金项目：国家重点研发计划（2018YFA0404603），国家自然科学基金委员会-中国科学院天文联合基金资助项目(U1931141,U1831204，U1631129)，云南省重点研发计划(2018IA054)，云南省应用基础研究项目(2017FB001, 2018FB103)，国家自然科学基金青年科学基金资助项目（11903009）资助。

作者：戴伟，男，副教授，汪森，男硕士研究生，昆明理工大学云南省计算机技术应用重点实验室。王锋，男，通信作者，教授，昆明理工大学云南省计算机技术应用重点实验室/广州大学，fengwang@gzhu.edu.cn

中的可用性。

二、研究动机与需求

2.1 研究动机

Spark 于 2009 年诞生于加州大学伯克利分校 AMPLab。已经成为 Apache 软件基金会旗下的顶级开源项目。相对于 MapReduce 上的批量计算、迭代型计算以及基于 Hive 的 SQL 查询，Spark 可以带来上百倍的性能提升。目前 Spark 的生态系统日趋完善，Spark SQL 的发布、Hive on Spark 项目的启动以及大量大数据公司对 Spark 全栈的支持，让 Spark 的数据分析范式更加丰富。Spark 与 Hadoop 的 MapReduce 计算框架类似，但相对 MapReduce 而言，Spark 特点更为突出，如其具有可伸缩、基于内存计算等特性，可以直接读写 Hadoop 上任何格式数据，在进行批处理时更加高效，延迟更低。目前已经成为轻量级大数据快速处理的统一平台。这其中，弹性分布式数据集(Resilient Distributed Datasets, RDD)是 Spark 的核心，RDD 是一种分布式的内存抽象，表示一个只读的记录分区的集合，

特别需要关注的是， Spark 是基于内存计算的大数据并行计算框架，提高了在大数据环境下数据处理的实时性，同时保证了高容错性和高可伸缩性，允许用户将 Spark 部署在大量廉价硬件之上，形成集群。这对于当前 SKA 科学数据处理来说，这样基于全内存与廉价集群的方式非常有吸引力。

算法参考库(ARL)[2]是由射电天文科学家 Tim Cornwell 领头开发的射电干涉阵数据处理算法验证库，用以为后续 SKA 的数据处理提供算法验证。目前，ARL 基于 Python 语言，已经实现了射电干涉阵处理的主要算法，全部程序开源，供射电干涉阵数据处理参考使用。自 2018 年 SKA 完成主要工作包的关键设计评估，进入桥接阶段后，ARL 已经成为系统学习、研究 SKA 数据处理的基础算法参考库。

针对 SKA 一期中频阵可见度校准问题，当前的 ARL 中，实现了一个全串行的 MID1 ICAL 管线，分为预测、校准、反馈、去卷积四个部分。MID1 ICAL 管线的特点如下：

(1) 逻辑任务同时具有数据密集型和计算密集型的特性；
(2) 逻辑任务之间存在数据依赖，且任务之间通讯量巨大；
(3) 需要多次迭代完成，并且两次迭代之间数据需要大量更新；
(4) 现有存储条件无法长时间保留原始数据，需要在一个观测周期内完成管线的执行。

为了满足 SKA 后续建设工作，如何将这样的串行代码移植到分布计算框架下，并分析其实现方法和性能变化，深入了解不同的算法在分布计算框架下的实现方式与性能评价，对于后续开展 SKA 科学数据处理有重要的作用，这也正是 SKA 桥接阶段的工作重点，需求非常迫切。

2.2 处理需求

SKA1 中频有 197 个天线，最大基线长度是 120 公里。SKA1 低频有 130000 个天线，最长基线长度 40 公里。预计 SKA 1 的数据注入速度约为 2TB/s。为进行实验与性能分析，本文中以 SKA 中频天线设计指标为基础，定义基线数 19306 个，81.92 个通道，每个通道宽度 800 MHz，4 个极化以及 36 个采样时间。总计算任务 28800 个。

在未来的系统部署中，成像管线（Imaging Pipeline）和非成像管线(Non-Imaging Pipeline)是其中的 2 个重点。其中成像管线将包括可见度函数接收、可见度函数预处理、实时校准、快速成像、瞬变源候选体检测、数据缓冲等多个部分。非成像管线将包括接收脉冲星守时特性文件、脉冲星候选体接收、瞬变观测缓冲、脉冲星守时处理等各个部分。从当前 SKA 的建设来看，这些管线的研制是整个数据处理系统的核心。本文讨论的管线，是实时

[2] https://github.com/SKA-ScienceDataProcessor/algorithm-reference-library

校准管线的一个关键部分。

三、软件实现与关键技术

3.1 代码基本流程

图 1 给出了 ICAL 中的管线单一通道对应的逻辑任务，将这样的逻辑任务由串行转为分布计算，最关键的是将可以原来紧耦合的功能，转变为松散耦合的模块。我们重点说明 Reppre_ifft 和 Degrid，以及 Pharotpre_dft_sumvis 两个功能部分。

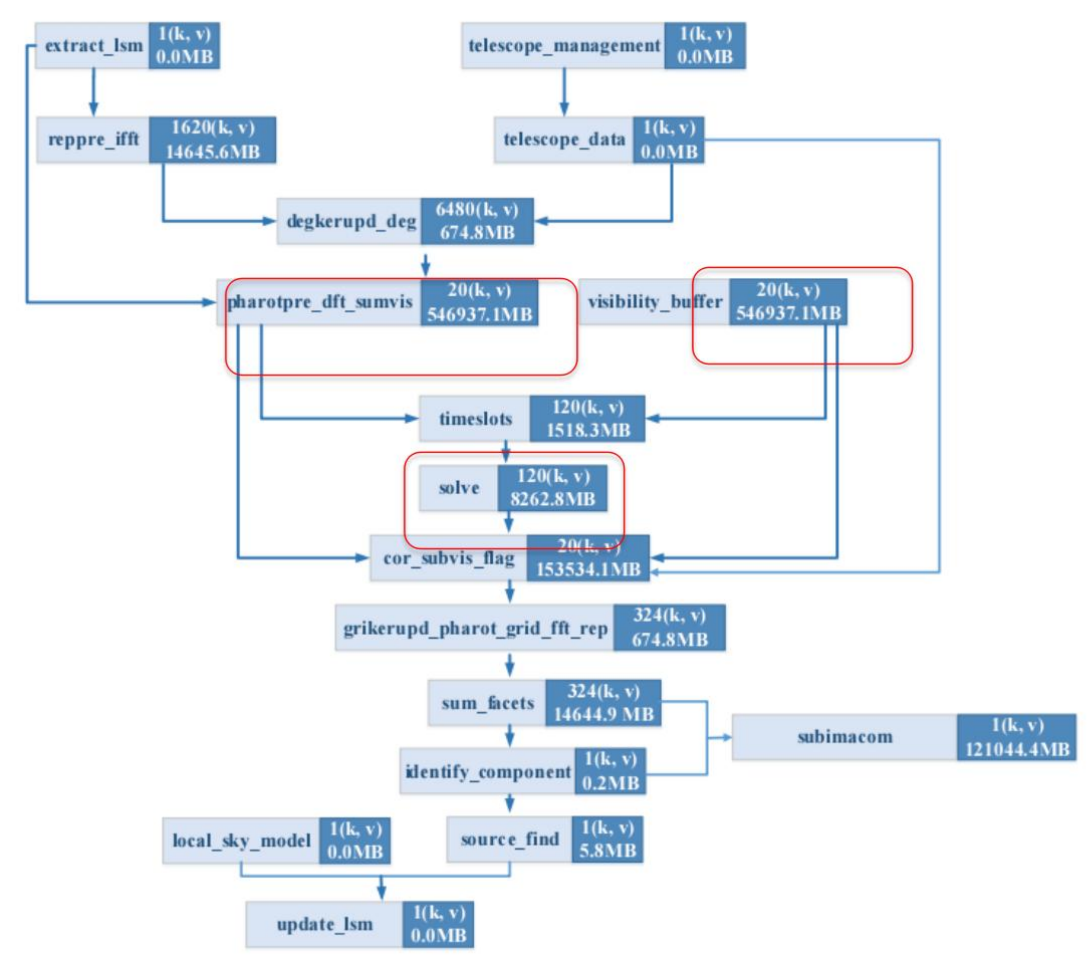


图 1. MID1 ICAL 管线单一通道对应的逻辑任务，括号内是需要处理的数据量

Figure 1. The logical diagram of MID1 ICAL Pipeline with single channel. The data in the parentheses shows the amount of data

3.1.1 Reppre_ifft 和 Degrid 代码实现

Reppre_ifft & Degrid 阶段是根据局部天空模型，预测观测天空的可见数据，开发中利用了如下 ARL 函数。对这一部分并行的实现方法是对频段、分片、时间片等进行数据的分片。最终对于 MID1 ICAL 管线，我们采用了以单个频段（不同频段可以并行处理）分拆任务的方法实现分布并行计算。

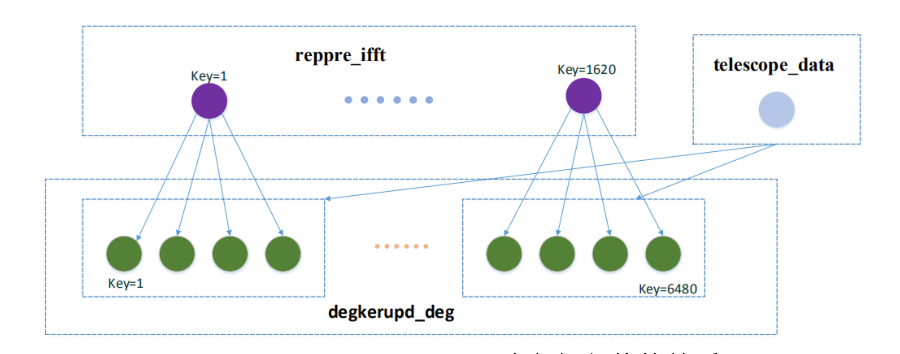


图 2　Degkerupd_deg 阶段任务依赖关系图

Figure 2. The dependency diagram in Degkerupd_deg phase

Reppre_fft 阶段主要是对 Image 类和 Skycomponent 的处理，整个过程的输入是 Image 和 Skycomponent，输出是 Image，并送入下个阶段。代码基本调用过程包括：

(1) arl/skycomponent/operations/insert_skycomponent: 将 skycomponent 中的信息按照 nchan 和 npol 两个轴插入到 Image，关联函数包括：insert_function_sinc, insert_function__L，insert_function_pswf，insert_function_array。

(2) arl/fourier_transforms/fft_support/fft: 对 Image 进行傅里叶变换

(3) arl/image/operations/reproject_image: 将切片后的 Image 按照新的 wcs 进行重投影。

(4) arl/image/iterators/raster_iter: Image 的切片函数

Degrid 阶段主要是对图像中的数据进行卷积，具体卷积的过程和可见度数据中的 uvw 等属性相关。并将结果放入本来为空的 Visibility 类的 Data 属性中。

在卷积之前，image 先被 padding 到切片之前的大小，即在它的周围填充 0。然后整个矩阵再和前一步在 get_kernelist 中得到的griding correction function 做一个点对点的乘法，对得到的结果再做一个 fft，就得到了通道化的 image，这个通道化的 image 接下来就将被卷积。

在分布执行后　，最终用以下方法合并 reppre_ifft 和 degrid 任务。

sc.parallelize(initset).flatMap(ix=>reppre_ifft_degrid_kernel(ix,broads_input_telescope_data,broadcast_lsm))

这里 initset 是六元组(beam, major_loop, frequency, time, facet, polarisation)。

2.1.1.2 Pharotpre_dft_sumvis

为了避免 RDD Pharotpre_dft_sumvis 的一个<key,value>对过大，我们将一个 item 中的时间分片个数由 120 减为 10．这样 RDD Pharotpre_dft_sumvis 的每一个<key,value>对中包括 20 个频率的 10 个时间片。图 3 给出了任务依赖关系图。

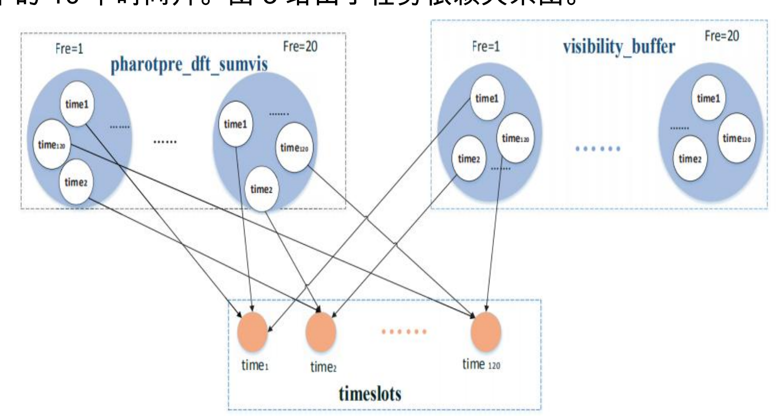


图 3. Timeslots 阶段任务依赖关系图

Figure 3. The dependency diagram in Timeslots phase

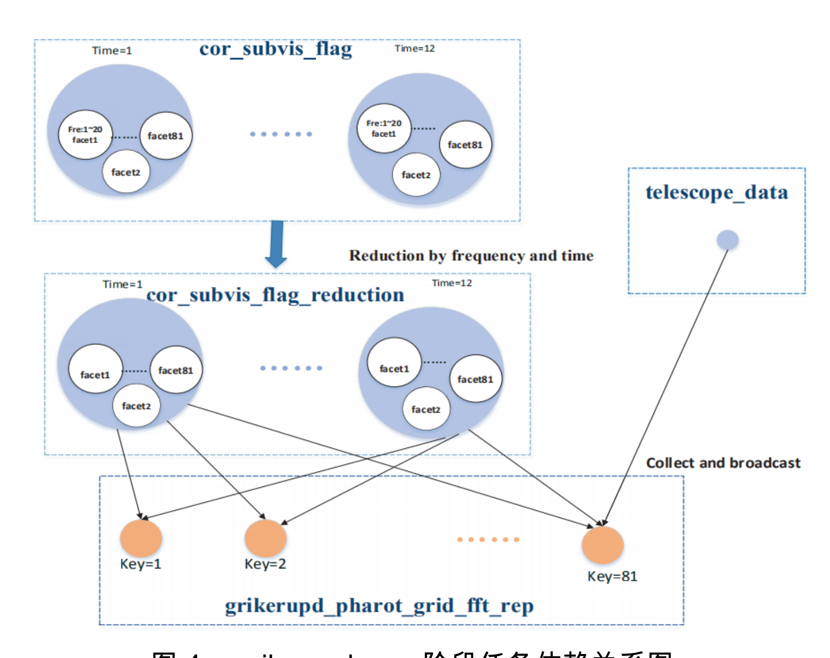


图 4. grikerupd_rep 阶段任务依赖关系图

Figure 4. The dependency diagram in grikerupd_rep phase.

SKA 管线的每个任务 IO 大小以及计算量都是确定的。对于 IO，输入、输出以及中间数据大小都是确定的，准确计算出每个任务具体的输入输出，以及准确计算出每个任务的计算量对分布式计算的任务调度具有指导意义。管线数据建模的依据是对每一个管线中的逻辑任务，通过分析所采用的算法的复杂度，并且计算输入输出的大小。

四、系统部署与测试

4.1 环境配置

在测试中我们采用三套不同配置的集群作为测试环境，集群 1 包括 1 个节点，该节点配备 1.5 TB 内存，CPU 为 80 核，每个核主频为 2.2 GHZ.。集群 2 包括 4 个节点，每个节点配备 64 GB 内存，CPU 为 8 核，每个核主频为 1.8 GHZ。软件系统中 Spark 的版本为 2.0，JDK 版本为 1.8 。

4.2 性能测试

在上述测试互不干涉中，我们分别对单机串行程序和基于 Spark 的程序进行了测试，串行程序完全是在集群 1 的高配环境中完成的，最终的执行时间见图 4 所示。

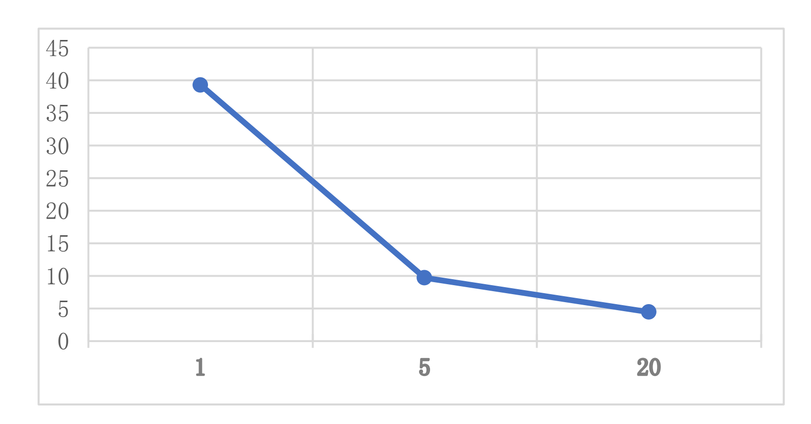


图 4 性能对比示意图，在 3 个进程数下执行的时间对比，Y 轴单位为秒
Figure 4. The contrast diagram of execution time elapse with 3 different process number, The unit of Y axis is second.

五、结论与未来工作

Spark 是当前工业界最流行的分布式执行框架之一。实验表明，基于 Spark 构建 SKA 的分布式执行框架是有可能性的，但 Spark 存在一些很实际的困难，具体说明如下：

1: Spark 考虑更多的是数据密集型，它的任务调度和资源管理目前只支持 CPU，需要更改其任务调度和资源管理代码来支持混合计算任务的调度。我们在研究中发现，Spark 性能瓶颈在于数据的连接操作，Spark 的“cogroup”需要对几个大的数据集进行排序操作，产生大量的节点通讯。另外 RDD 链过长，内存不能及时释放，Spark 内存不足时，数据需要写到磁盘，序列化与反序列化耗费大量时间。Spark cogroup 和 groupByKeys 需要排序操作，会耗费大量内存。从这个方面来看，Spark 要满足 SKA 的建设要求存在较多待改进的地方。在未来可以值得在如下方面继续进行性能优化

1）增大内存。Spark 内存不足时，数据需要写到磁盘，序列化与反序列化耗费大量时间。Spark 的 cogroup 操作需要内存排序，耗费内存资源。

2）利用分布式缓存系统替换 Cogroup 操作；

3）利用分布式缓存系统存储部分 RDD 的内容，打破长 RDD 链，及时释放内存资源；

4）利用 Spark 的 Partitioning 操作代替 cogroup, Partitioning 可以做到数据重组。同时设计分区函数，较少 join 操作。根据射电数据处理的特点，在预测阶段，设计根据频率的分区函数，需要将同一频段的可见度数据聚集在一起，按照频段进行分区，可以避免 join 操作。在去卷积阶段，设计根据分片的分区函数，避免 join 操作。

2: Spark 的 shuffle 性能存在严重缺陷，Shuffle 是 MapReduce 框架中的一个特定的 phase(分阶段),介于 Map phase 和 Reduce phase 之间 ,当 Map 的输出结果要被 Reduce 使用时 ,输出结果需要按 key 哈希 ,并且分发到每一个 Reducer 上去 ,这个过程就是 shuffle。由于 shuffle 涉及到了磁盘的读写和网络的传输 ,因此 shuffle 性能的高低直接影响到了整个程序的运行效率。针对天文海量数据处理这样的要求，Spark 的 shuffle 性能显然满足不了要求。我们拟采用内存数据库来代替 shuffle 的相关操作，并可以进一步提高 Spark 的并行计算性能。这样的工作是后续研究的重点。

参考文献：

[1] Carilli C, Rawlings S. Science with the Square Kilometer Array: motivation, key science

projects, standards and assumptions [J]. arXiv preprint astro-ph/0409274, 2004,

[2] Dewdney P, Lazio T, Hall P, et al. The square kilometer array (SKA) radio telescope: Progress and technical directions [J]. Radio Science Bulletin, 2008, 326(4-19.

[3] Staveley-Smith L. The Square Kilometre Array [J]. Australasian Science, 2009,

[4] Broekema P C, Nieuwpoort R V V, Bal H E. ExaScale high performance computing in the square kilometer array; proceedings of the The Workshop on High-Performance Computing for Astronomy DATE, F, 2012 [C].

[5] 赖铖，梅盈，邓辉, et al. MUSER 可见度数据积分方法与实现 [J]. 天文研究与技术, v.15;No.57(1): 81-9.

[6] 于晓雨，邓辉，梅盈, et al. 宽视场成像网格化算法中 w-plane 最优经验值研究 [J]. 天文研究与技术, 2019, 16(2): 218-24.

[7] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster computing with working sets [J]. HotCloud, 2010, 10(10-10): 95.

[8] Spark A. Apache Spark: Lightning-fast cluster computing [M]. 2015.

[9] Rocklin M. Dask: Parallel computation with blocked algorithms and task scheduling; proceedings of the Proceedings of the 14th python in science conference, F, 2015 [C]. Citeseer.

Implementation of SKA1-MID self-calibrating pipeline based on Spark

Dai Wei[1] Wang Sen[1] Li Qiuhong[2] Deng Hui[3] Mei Ying[3] Wang Feng [123]*

1. Yunnan Key Laboratory of Computer Technology Application, Kunming University of Science and Technology, Kunming, Yunnan, 650051
2. Fudan University, Shanghai, 210000
3. Astrophysics Center of Guangzhou University, Guangzhou 510006, China

Abstract

The amount of the scientific data generated by the SKA exceeds the processing capabilities of all existing distributed processing systems. How to implement a distributed execution framework is an important research issue of scientific data processing. Based on Spark framework, oen of the the most mature execution frameworks, this study attempts to systematically analyze how to migrate iCal pipelines in the Algorithm Reference Library (ARL) to Spark. We analyze and discusses the implementation procedure and present the corresponding task flow implementation. The final experiments show that the results of the iCAL upon Spark is correct. In summary, Spark could meet the requirements of distributed data for certain data. The limitations of Spark itself severely restrict its application in SKA.

Keywords: Distributed Pipeline, Spark, ARL, Distributed data processing